Programmation Fonctionnelle Avancée

Séance 2 : Fold de listes, map & fold des arbres

Alexandros Singh

Université Paris 8

12 octobre 2023

Problème : Somme d'une liste

Étant donné une liste `l` d'entiers naturels, retourner la somme de ses éléments.

Problème : Somme d'une liste

Étant donné une liste `l` d'entiers naturels, retourner la somme de ses éléments.

```
let rec sum_list = function
    [] -> 0
    | x :: xs -> x + sum_list(xs)
```

Problème : Produit d'une liste

Étant donné une liste `l` d'entiers naturels, retourner le produit de ses éléments.

Problème : Produit d'une liste

Étant donné une liste `l` d'entiers naturels, retourner le produit de ses éléments.

```
let rec sum_list = function
    [] -> 1
    | x :: xs -> x * prod_list(xs)
```

Problème : Recherche dans une liste

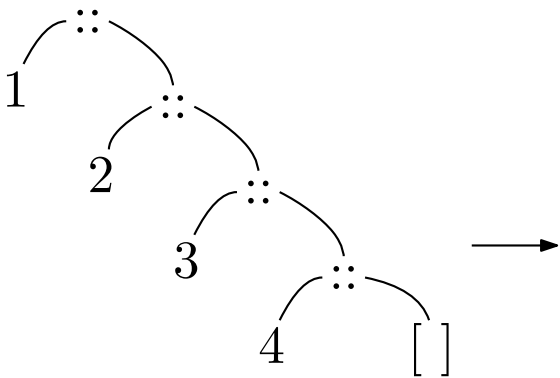
Étant donné une liste `l` d'entiers naturels et un entier naturel `e`, retourner vrai si la `l` contient `e`, sinon retourner faux.

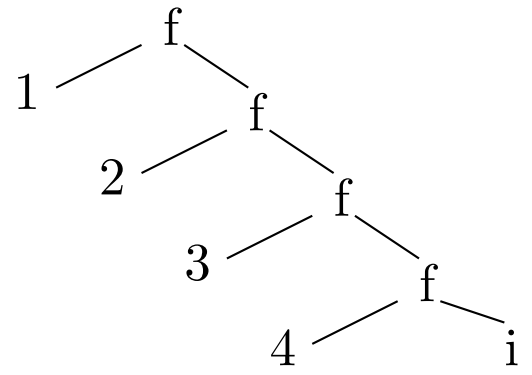
Problème : Recherche d'une liste

Étant donné une liste `l` d'entiers naturels et un entier naturel `e`, retourner vrai si la `l` contient `e`, sinon retourner faux.

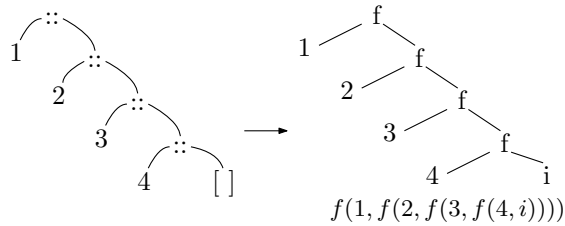
```
let rec search_in_list = function
    [], _ -> false
    | h :: t, e -> (h = e) || (search_in_list (t,e))
```

Fold



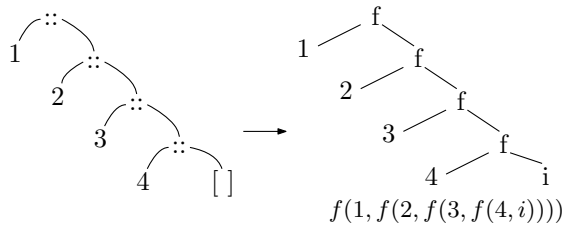


Fold




```
let rec my_fold_right (f : 'a -> 'b -> 'b) (i : 'b)
(l : 'a list) : 'b = match l with
     [] -> i
   | h :: t -> f h (my_fold_right f i t)
```

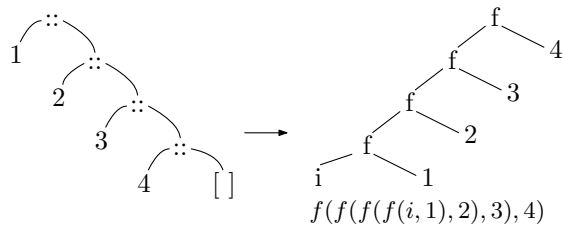
Fold




```
let rec my_fold_right (f : 'a -> 'b -> 'b) (i : 'b)
(l : 'a list) : 'b = match l with
     [] -> i
   | h :: t -> f h (my_fold_right f i t)
```

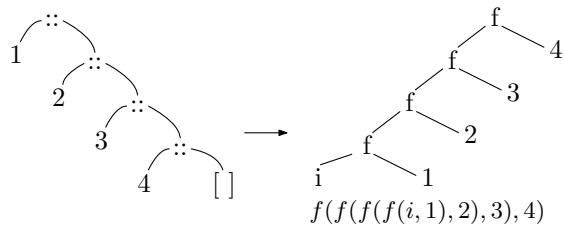
Pas de récursivité terminale ! Pouvez-vous penser à un récursive terminale ?

Fold



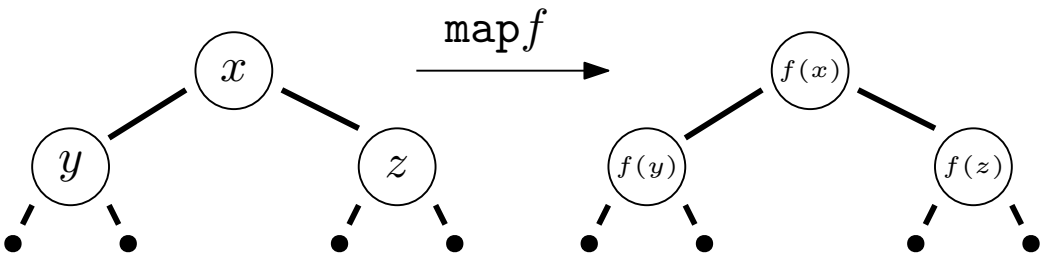

```
let rec my_fold_left (f : 'b -> 'a -> 'b) (i : 'b)
(l : 'a list) : 'b = match l with
    | [] -> i
    | h :: t -> my_fold_left f (f i h) t
```

Fold

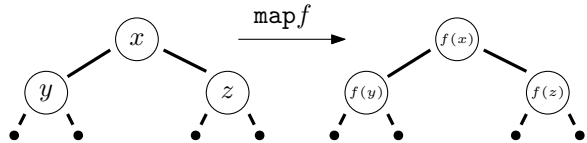



```
let rec my_fold_left (f : 'b -> 'a -> 'b) (i : 'b)
(l : 'a list) : 'b = match l with
    | [] -> i
    | h :: t -> my_fold_left f (f i h) t
```

La fonction `map` se généralise immédiatement au cas des arbres binaires :

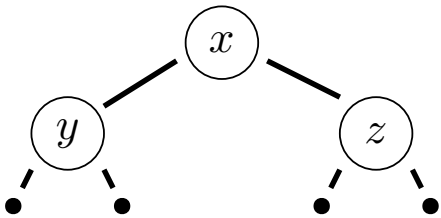


La fonction `map` se généralise immédiatement au cas des arbres binaires :



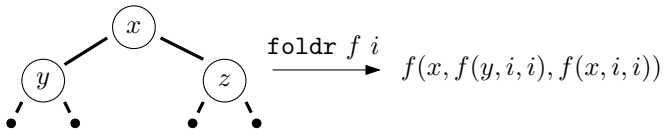

```
let rec map_bin_tree f = function
    Leaf -> Leaf
    | Node (v,l,r) -> Node (f v, map_bin_tree f l, map_bin_tree f r)
```

De même, la fonction `foldr` sur les arbres donne :



$$\texttt{foldr}\ f\ i \longrightarrow f(x, f(y, i, i), f(x, i, i))$$

De même, la fonction `foldr` sur les arbres donne :




```
let rec foldr_bin_tree f i = function
    Leaf -> i
    | Node (v,l,r) -> f v (foldr_bin_tree f i l) (foldr_bin_tree f i r)
```